JESD3-C

# JEDEC STANDARD

# Standard Data Transfer Format Between Data Preparation System and Programmable Logic Device Programmer

## JESD3-C

**(Revision of JESD3-B)**

**JUNE 1994**

ELECTRONIC INDUSTRIES ASSOCIATION
ENGINEERING DEPARTMENT

NOTICE

JEDEC Standards and Publications contain material that has been prepared, progressively reviewed, and approved through the JEDEC Council level and subsequently reviewed and approved by the EIA General Counsel.

JEDEC Standards and Publications are designed to serve the public interest through eliminating misunderstandings between manufacturers and purchases, facilitating interchangeability and improvement of products, and assisting the purchaser is selecting and obtaining with minimum delay the proper product for his particular need. Existence of such standards shall not in any respect preclude any member or nonmember of JEDEC from manufacturing or selling products not conforming to such standards, nor shall the existence of such standards preclude their voluntary use by those other than EIA members, whether the standard is to be used either domestically or internationally.

JEDEC Standards and Publications are adopted without regard to whether their adoption may involve patents or articles, materials, or processes. By such action, JEDEC does not assume any liability to any patent owner, nor does it assume any obligation whatever to parties adopting the JEDEC Standards or Publications.

The information included in JEDEC Standards and Publications represents a sound approach to product specification and application, principally from the solid state device manufacturer viewpoint. Within the JEDEC organization there are procedures whereby a JEDEC Standard or Publication may be further processed and ultimately became an EIA Standard.

Inquiries, comments, and suggestions relative to the content of this JEDEC Standard should be addressed to the JEDEC Executive Secretary at EIA Headquarters, 2001 Pennsylvania Ave., N.W., Washington, D.C. 20006.

Published by

©ELECTRONIC INDUSTRIES ASSOCIATION 1994
Engineering Department
2001 Pennsylvania Ave., N.W.
Washington, D.C. 20006

**PRICE: Please refer to the current**
**Catalog of EIA, JEDEC, and TIA STANDARDS and ENGINEERING PUBLICATIONS**
**or call Global Engineering Documents, USA and Canada (1-800-854-7179)**
**International (303-397-7956)**

Printed in U.S.A.

**PLEASE!**

**DON'T VIOLATE**
**THE**
**LAW!**



EIA Engineering Publications Office
2001 Pennsylvania Ave., N.W.
Washington, D.C. 20006
(202)457-4963

# STANDARD DATA TRANSFER FORMAT BETWEEN DATA PREPARATION SYSTEM AND PROGRAMMABLE LOGIC DEVICE PROGRAMMER

## CONTENTS

# 1 INTRODUCTION

## 1.1 Purpose and Scope

This standard was developed to prevent the proliferation of data transfer formats that occurred with microprocessor development systems. The focus of the standard is on field-programmable devices and their support tools. It is not intended for other types of semicustom logic devices or other types of fabrication or testing equipment.

The standard includes a simple transmission protocol based on traditional PROM formats that allow a device programmer to share a computer serial port with a terminal. This simple protocol is not a complete communication protocol and does not do retries or error correction. If the device programmer has local storage, such as floppy disk, for the programming data this protocol is not required.

This standard defines a data format for transferring the fuse or cell states between the development system and the programmer. It does not define the device architecture such as types of logic arrays or the output macro cell. Also the standard does not define the programming algorithms or the device and technology specific information for accessing the fuses or cells.

Field-programmable logic devices may require more testing than programmable memories, so the standard defines a simple functional testing format. This test vector format is not a general purpose parametric test language.

```
<STX>File for PLD 12S8  Created on 8-Feb-85  3:05PM
6809 memory decode  123-0017-001
Joe Engineer  Advanced Logic Corp *
QP20* QF448* QV8*
F0* X0*
L0000 11111011111111111111111111111*
L0028 10111111111111111111111111111*
L0056 11101111111111111111111111111*
L0112 01010111011110111111111111111*
L0224 01010111101110111111111111111*
L0336 01010111011101111111111111111*
V0001 000000XXXNXXXHHHLXXN*
V0002 010000XXXNXXXHHHLXXN*
V0003 100000XXXNXXXHHHLXXN*
V0004 110000XXXNXXXHHHLXXN*
V0005 111000XXXNXXXHLHHXXN*
V0006 111010XXXNXXXHHHHXXN*
V0007 111100XXXNXXXHHLHXXN*
V0008 111110XXXNXXXLHHHXXN*
C124E*<ETX>8A76
```

**Figure 1. Example of a Programmable Logic Device Data File**

### 1.2 Summary of Programming and Testing Fields

The programming and testing information is contained in various fields. The following list gives the field identifier and a description. To comply with the standard the device programmer, tester, and development system must provide and recognize certain fields.

| Identifier | Description |
|---|---|
| (n.a.) | Design Specification |
| N | Note |
| QF | Number of Fuses in Device *** |
| QP | Number of Device Package Pins *** |
| QV | Maximum Number of Test Vectors *** |

| | | | |
|---|---|---|---|
| F | Default Fuse State * | E | Electrical Data * |
| L | Fuse List * | U | User Data |
| C | Fuse Checksum | J | Device Identification |

| | |
|---|---|
| X | Default Test Condition ** |
| V | Test Vectors ** |
| P | Pin Sequence ** |

| | |
|---|---|
| D | Device (Obsolete) |
| G | Security Fuse |
| R,S,T | Signature Analysis |
| A | Access Time |

| | |
|---|---|
| * | Programmer must recognize |
| ** | Tester must recognize |
| *** | Development System must provide |

### 1.3 Changes to October 1983 Standard

The 1983 standard defined the D, F, L, C, V, and P field. Several other fields are now being used, and this new standard formally defines these fields. The new standard is a superset of the 1983 standard.

The sequence and allowed combinations of the various fields have been clarified. One field, the D field, was not clearly defined and this led to conflicting uses. The D field is made obsolete in this standard.

Several changes have been added to the test vectors to defined unspecified vectors and don't care conditions. The test vectors now allow for manufacture independent register preloading.

### 1.4 Changes clarifying preload test vectors

The B preload test vector, intended to allow preloading of buried registers, has been clarified. The previous definition was ambiguous. In addition, register numbering guidelines have been added, and a "read-and-retain" test condition has been added to allow selective preloading of registers, without disturbing other registers.

In Standard 3-A, the QP field was defined as containing the number of pins in the test vectors. The new standard defines the QP field as containing the number of pins on the device package. Conditions under which the QP field is mandatory are listed in 5.4

### 1.5 Addition of a register observation vector

A T vector has been added to allow the observation of all registers in a device. This allows equal access to output, input, and internal registers. Its format parallels that of the B preload vector.

### 1.6 Additions to JESD3-B which implement JESD3-C

Electrical Fuse Data (E): The E field has been added to allow for the handling of special feature fuses that do not affect the fucntion of the device. An example of this type of fuse is the power miser fuses available in some types of PLD devices. With this addition, the integrity of the L field will remain intact such that one common JEDEC file can be utilized to program any device with the same logical functionality.

User Data (U): The U field has been added to allow for the handling of special feature fuses that do not affect the logic function of the device. An example of this type of fuse is the User Data Signature (information only) available in some types of PLD devices. With this addition, the integrity of the L field will remain intact such that one common JEDEC file can be utilized to program any device with the same logical functionality.

Device Identification (J): The purpose of the J field is to provide a means of verifying that a particular JEDEC file is appropriate for the device which has been selected for programming. Each PLD architecture is assigned a unique code, and each pinout variation of an architecture is assigned a unique code. Uniqueness is defined more thoroughly below. The intent is for the device programmer to check the device and JEDEC file before programming and then issue an error message if the device code in the JEDEC file does not match the device type that has been selected for programming. This field does not provide for verification of the actual device in the socket.

## 2 SPECIAL NOTATIONS AND DEFINITIONS

### 2.1 Notation Conventions

In addition to descriptions and examples this document uses the Backus-Naur Form (BNF) to define the syntax of the data transfer format. BNF is a shorthand notation that follows these rules:

- "::=" denotes "is defined as".
- Characters enclosed by single quotes are literals (required).
- Angle brackets enclose identifiers.
- Square brackets enclose optional items.
- Braces (curly brackets) enclose a repeated item. The item may appear zero or more times.
- Vertical bars indicate a choice between items.
- Repeat counts are given by a :n suffix. For example, a six digit number would be defined as "<number> ::= <digit>:6."

For example, in words, the definition of a person's name reads:

> The full name consists of an optional title followed by a first name, a middle name, and a last name. The person may not have a middle name or may have several middle names. The titles consist of: Mr., Mrs., Ms., Miss, and Dr.

BNF Syntax:

<full name> ::= [<title>] <f. name> {<m. name>} <l. name>
<title> ::= 'Mr.' | 'Mrs.' | 'Ms.' | 'Miss' | 'Dr.'

Examples

Miss Mary Ann Smith

Mr. John Jacob Joseph Jones

Tom Anderson

## 2.2 BNF Rules and Definitions

The following standard definitions are used throughout the rest of this document:

<digit> ::= '0' | '1' | '2' | '3' | '4'
| '5' | '6' | '7' | '8' | '9'

<hex-digit> ::= <digit>
| 'A' | 'B' | 'C' | 'D' | 'E' | 'F'

<binary-digit> ::= '0' | '1'

<number> ::= <digit> {<digit>}

<del> ::= <space> | <carriage return>

<delimiter> ::= <del> {<del>}

<printable character> ::= <ASCII 20 hex ... 7E hex>

<control character> ::= <ASCII 00 hex ... 1F hex>
| <ASCII 7F hex>

<STX> ::= <ASCII 02 hex>
<ETX> ::= <ASCII 03 hex>
<carriage return> ::= <ASCII 0D hex>
<line feed> ::= <ASCII 0A hex>
<space> ::= <ASCII 20 hex> | ' '

<valid character> ::= <printable character>
| <carriage return> | <line feed>

<field character> ::= <ASCII 20 hex ... 29 hex>
| <ASCII 2B hex ... 7E hex>
| <carriage return> | <line feed>

### 2.3 PLD Register numbering

The B and T vectors require that PLD manufacturers provide a register grouping and numbering sequence to the PLD programmer vendors. This numbering sequence will be used in the B and T vectors.

The registers are numbered from 1 to N, where N is the maximum number of registers in the device. The PLD manufacturer is responsible for assigning and documenting the register numbers.

There are three types of registers: output, internal, and input. Output registers are connected directly to a device pin used as an output. Internal registers do not have a direct connection (or any connection) to a device pin. Input registers are connected to device pins used as dedicated inputs. If a register can be used as an input or output register, then this register is classified as an output register for register numbering purposes.

Registers must be sequenced in the following order:

1. Output and I/O registers
2. Internal registers
3. Input Registers

Within a group, all registers should be numbered in ascending sequence, in the order of the lowest-number programmable element that can be associated with each register. For example, if the lowest numbered element used by internal register A is 31 and the lowest-numbered element used by internal register B is 61, than A is numbered first in the register sequence. However, even if the lowest-numbered element used by output register C is 91, C is still numbered ahead of A and B, since output registers are numbered before internal registers.

## 3 TRANSMISSION PROTOCOL

### 3.1 Protocol Syntax

This simple STX-ETX protocol is based on traditional PROM formats that allow a device programmer to share a serial computer port with a terminal. The transmission consists of a start-of-text (STX) character, various fields, an end-of-text (ETX) character, and a transmission checksum. The character set consists of the printable ASCII characters and four control characters (STX, ETX, CR, LF). Other control characters should not be used because they can produce undesirable side-effects in the receiving equipment.

Syntax of the Transmission Protocol:

<format> ::= <STX> {<field>} <ETX> <xmit checksum>

### 3.2 Computing the Transmission Checksum

The transmission checksum is the 16-bit sum (i.e., modulo 65,535) of all ASCII characters transmitted between and including the STX and ETX (see figure 2). The parity bit is excluded in the calculation.

Syntax of the Transmission Checksum:

<xmit checksum> ::= <hex-digit>:4

random text <return><line feed> = 0000
<STX>TEST*<return><line feed> 02+54+45+53+54+2A+0D+0A = 0183
QF0384*<return><line feed> 51+46+30+33+38+34+2A+0D+0A = 01A7
F0* <return><line feed> 46+30+2A+20+20+0D+0A = 00F7
L10 101*<return><line feed> 4C+31+30+20+31+30+31+2A+0D+0A = 01A0
<ETX>05C4 <return> random text 03 = 0003

05C4

**Figure 2. Computing the Transmission Checksum.**

### 3.3 Disabling the Transmission Checksum

Some computer operating systems do not allow the user to control what characters are sent, especially at the end of a line. The receiving equipment should always accept a dummy value of "0000" as a valid checksum. This dummy checksum is a method of disabling the transmission checksum.

## 4 DATA FIELDS

### 4.1 General Field Syntax

In general, each field in the format starts with an identifier, followed by the information, and terminated with an asterisk. For example, "C1234*" specifies that the checksum of the fuse data is 1234. The design specification header does not have an identifier and must be the first field in the transmission, immediately following the STX.

Syntax of Fields:

<field> ::= [<delimiter>] <field identifier>
{<field character>} '*'

<field identifier> ::= 'A' | 'C' | 'D' | 'E' | 'F' | 'G' | 'J'
| 'L' | 'N' | 'P' | 'Q' | 'R'
| 'S' | 'T' | 'U' | 'V' | 'X'

<reserved identifier> ::= 'B' | 'H' | 'I' | 'K'
| 'M' | 'O' | 'W' | 'Y' | 'Z'

### 4.2 Field Identifiers

Each field begins with a single character identifier that identifies the field type. Multiple character identifiers can be used to create subfields (i.e., "A1", "A$", or "AB3"). The field is terminated with an asterisk. Therefore, asterisks cannot be embedded within the field. While not required, carriage returns and line feeds should be used to improve the readability of the format. Reserved identifiers currently have no function and are reserved for future use. Receiving equipment should ignore fields starting with reserved identifiers. The meanings of the field identifiers are given in table 1.

| | |
|---|---|
| A - Access Time | N - Note |
| B - * | O - * |
| C - Checksum | P - Pin sequence |
| D - Device type | Q - Value |
| E - Electrical Fuse Data | R - Resulting vector |
| F - Default fuse state | S - Starting vector |
| G - Security fuse | T - Test cycles |
| | U - User Data |
| I - * | V - Test vector |
| J - Device Identification | W - * |
| K - * | X - Default test condition |
| L - Fuse list | Y - * |
| M - * | Z - * |

**Table 1. Field Identifiers (* indicates reserved for future use)**

## 5 COMMENT AND DEFINITION FIELDS

### 5.1 Design Specification

The design specification is the first field in the format, must be included, and does not have an identifier signaling its start. An asterisk terminates the field. The contents of the design specification are not defined but should consist of

1. User's name and company
2. Date, part number, and revision
3. Manufacturer's device number
4. Other information

Syntax of the Design Specification:

<design specification> ::= {<field character>} '*'

Example:

File for PLD 12S8
Created on 8-Feb-85 3:05PM
6809 memory decode 123-0017-001
Joe Engineer Advanced Logic Corp *

A blank field consisting of the terminating asterisk is valid design specification field.

Example:

*

### 5.2 Note (N)

The note field is used to place notes and comments in the data file. The note field(s) may appear anywhere in the file and the receiving equipment may ignore this field.

Syntax of the Note Field:
<note> ::= 'N' <field characters> '*'

Example:
N Following vectors were modified for the ACME 123 tester*

### 5.3 Device Definition (D) Obsolete

This field is now obsolete; it has been eliminated to ensure the format is device and technology independent.

### 5.4 Values (QF, QP, QV)

The Q field expresses values or limits that must be provided to the receiving equipment. Three subfields are defined: the F subfield for the number of fuses, the P subfield for number of pins or test conditions in the test vector, and the V subfield for the maximum number of test vectors.

These values enable the receiving device to efficiently allocate memory and perform certain calculations. The QF field tells the receiving equipment how much memory to reserve for fuse data, the number of fuses to set to the default condition, and the number of fuses to include in the fuse checksum.

The QP field tells the receiving equipment the number of physical package pins on the device. This field is mandatory if the following three conditions exist:

a) non connected pins exist on the device (such as a 24-pin device packaged in a 28-pin SCC), and
b) the file contains test vectors, and
c) the programming equipment does not automatically handle the translation required to accommodate the non connected package pins.

The value fields must occur before any device programming or testing fields in the data file. Files with only testing fields do not require the QF field and files with only programming fields do not require the QP and QV fields. The QP field must specify all device pins for files that contain B preload vectors.

Syntax for Value Fields:
<fuse limit> ::= 'QF' <number> '*'
<number of pins> ::= 'QP' <number> '*'
<vector limit> ::= 'QV' <number> '*'

Example:
QF1024* Indicates device has 1024 fuses
QP24* Indicates 24 pins on device package
QV250* Indicates a maximum of 250 test vectors

Table 2. Test Condition

# 6 DEVICE PROGRAMMING FIELDS

## 6.1 Syntax and Overview

Each fuse or cell of a device is assigned a decimal number and has two possible states: a zero, specifying a low resistance link (a logical connection between two points); or a one, specifying a high resistance link (no logical connection between two points). The fuse numbers start at zero and are consecutive to the maximum fuse number. For example, a device with 2048 fuses would have fuse numbers between 0 and 2047. Fuse information describing the state of each fuse in the device is given by three fields: the default state (F field), the fuse list (L field), and the fuse checksum (C field).

All user programmable fuses or cells may be specified with a L field. There are no separate fields for control terms or architecture fuses.

Syntax of the Fuse Information fields:

<fuse information> ::= [<default state>] <fuse list>
{<fuse list>} [<fuse checksum>]

<default state> ::= 'F' <binary-digit> '*'

<fuse list> ::= 'L' <number> <delimiter>
{<binary-digit> [<delimiter>]} '*'

<fuse checksum> ::= 'C' <hex-digit>:4 '*'

Example:

F0*
L0000 01001110 00001000 11110000 11111111 01010001*
C021A*

## 6.2 Fuse Default State (F)

The F field defines the state of fuses that are not explicitly defined in the L field. If no F field is specified, all fuse states must be defined by L fields. If the default state is used, it must be specified after the QF field and before the first L field.
Example:

F0* Set default to 0

### 6.3 Fuse List (L)

The L field starts with a decimal fuse number and is followed by a stream of fuse states (0 and 1). The fuse number may include leading zeros (i.e. "L12" and "L0012" are the same). A space and/or a carriage return must separate the fuse number from the fuse states. The stream of fuse states can be as long as desired (up to the maximum allowable fuse number).

If the state for a fuse is specified more than once, the last state replaces all previous ones specified for that fuse. This allows a file to be modified or "patched" by appending new fuse states to the file.

Example:

```
L0000
11111011111111111111111111
10111111111111111111111111
11101111111111111111111111
00000000000000000000000000*
```

Example:

```
L0000
1111101111111111111111111110111111111
111111111111111
11101111111
111111111111111
00000000000000000000000000*
```

Example:

```
L00 11111011111111111111111111*
L28 10111111111111111111111111*
L56 11101111111111111111111111*
L84 00000000000000000000000000*
```

### 6.4 Fuse Checksum (C)

The fuse information checksum field is used to detect transmitting and receiving errors. The checksum is for the entire device (fuse number 0 to maximum fuse number set by the QF field), not just the fuse states sent. If multiple C fields are received only the last is significant.

The field contains the 16-bit sum (i.e., modulo 65,535) of the 8-bit words containing the fuse states for the entire device. The 8-bit words are formed as shown in figure 2. Unused bits in the final 8-bit word are set to zero before the checksum is calculated.

```
word 00  |msb| | | | | | |lsb|
Fuse No.  7 6 5 4 3 2 1 0

word 01  |msb| | | | | | |lsb|
Fuse No. 15 14 13 12 11 10 9 8
                  ---------------
word 62  |msb| | | | | | |lsb|
Fuse No.  - - - - 499 498 497 496
```

**Figure 2. 8-Bit Words formed from fuse states for Checksum**

```
QF500*
F0*
L0000 01001110 00001000 11110000 11111111 01010001*
C021A*
```

| Fuse Number | MSB LSB | |
|---|---|---|
| 0000 | 0 1 1 1 0 0 1 0 | 72 |
| 0008 | 0 0 0 1 0 0 0 0 | 10 |
| 0016 | 0 0 0 0 1 1 1 1 | 0F |
| 0024 | 1 1 1 1 1 1 1 1 | FF |
| 0032 | 1 0 0 0 1 0 1 0 | 8A |
| 0040 | 0 0 0 0 0 0 0 0 | 00 |
| 0048 | 0 0 0 0 0 0 0 0 | 00 |
| 0488 | 0 0 0 0 0 0 0 0 | 00 |
| 0496 | - - - - 0 0 0 0 | 00 |
| | Fuse Checksum | 021A |

**Figure 3. Computing the fuse checksum.**

### 6.5 Electical Fuse Data (E)

The E field allows special feature fuses that do not affect the logic function of the device to be added without impact to existing JEDEC files for that device. An example of this type of fuse is the power miser fuse available in some types of PLD devices. With this addition, the integrity of the L field will remain intact such that one common JEDEC file can be utilized to program any device with the same logical functionality.

There are two types of E fields depending on how the data is entered. The E field is for binary data, and the EH field is for Hex data.

Syntax of the E field:

<Electrical DATA FUSE LIST>::'E'<binary digit>'*'
'EH'<hex digit>'*'

Examples:

NO ELECTRICAL FUSE DATA

```
QF24*
L0000
10101100000000000000000000*
C00AC*
```

ELECTRICAL FUSE DATA PRESENT

```
QF24*
L0000
10101100000000000000000000*
C00AC*
E10100111*
```

The E and H fields are read left to right from MSB to LSB.

Example:

```
Binary  E11001010*
or
Hex     EHCA*
```

### 6.6 User Data (U)

The U field allows user data fuses that do not affect the logical or electrical functionality of the device to be added without impact to existing JEDEC files for that device. An example of this type of fuse is the User Data Signature (information only) available in some types of PLD devices. With this addition, the integrity of the L field will remain intact such that one common JEDEC file can be utilized to program any device with the same logical functionality.

There are three types of U fields depending on how the data is entered. The U field is for binary data, the UA field is for ASCII field characters, and the UH field is for Hex data.

Syntax of the U field:

<User DATA FUSE LIST>::'U'<binary digit>'*'
'UA'<field character>'*'
'UH'<hex digit>'*'

The U, UH, and UA fields read left to right from MSB to LSB. If the field defined is larger than the available space in the device, the most significant bits should be truncated by the programmer until the appropriate size is reached. If the field is smaller than the available space in the device, the data will be fit into the device by the programmer beginning with the least significant bit. The unused bits will be filled with zeros by the programmer. There will be some binary combinations that cannot be represented as a field character in the UA field. The field character(s) in the UA field are assumed to be 7 bit quantities. U field data is not to be included in the C field fuse checksum.

Example: File with 28 user fuses

Binary U10101001000101101100010101 00*
or
Hex UHA916C54*
or
ASCII UATEXT*

Examples:

NO USER DATA FUSES

```
QF24*
L0000
10101100000000000000000000*
C0035*
```

USER DATA FUSES PRESENT

```
QF24*
L0000
10101100000000000000000000*
C0035*
UATEST
MSB LSB
MSBYTE LSBYTE
```

### 6.7 DEVICE IDENTIFICATION (J)

The J field provides a device identification code that uniquely specifies the logical architecture of the device for which the file is intended. The field consists of two decimal numbers seperated by a space and is terminated with an asterisk (*). The first number is an architecture code; the second number is a pinout code. The field is placed in front of any F, L, or H fields. The J field is optional.

Syntax for JEDEC device identification field:

<device code>::=J<number><space><number>*

The numbers in the J field are decimal integers greater than or equal to zero (0) as allowed below. The actual number for any given device is assigned by JEDEC as defined below.

The special pinout code zero (0) will be used to indicate that for a given JEDEC file, no particalar pinout is to be enforced. In this case, only the architecture code will be used to determine the appropriateness of the JEDEC file for the device selected.

Maximum protection is provided to the user by the combination of the architecture and pinout codes.

In no way is this code to be construed as an endorsement by JEDEC of any architecture or pinout. Any device can be given a code, and no architecture or pinout standardization is implied.

The number used in the J field is assigned by JEDEC. Contact the JEDEC office for the request of a Device Code form and details on assignment procedures.

A code assignment application fee will be charged by the JEDEC office. An invoice will be sent to the Sponsor, according to a fee schedule approved by the JEDEC Solid State Products Council. The fee schedule may recognize a group assignment for a family of devices that are receiving codes at the same time.

## 7 DEVICE TESTING FIELDS

### 7.1 Syntax and Overview

Functional test information is specified by test vectors containing test conditions for each device pin.

Syntax of Functional Test Information:

<function test> ::= [<default test condition>]
[<pin list>] <test vector>
{<test vector>}

<default test condition> ::= 'X' <binary digit> '*'

<pin list> ::= 'P' <pin number>:N '*'

<pin number> ::= <delimiter> <number>

N ::= number of pins on device

<test vector> ::= 'V' <number> <delimiter>
<test condition>:N '*'

<test condition> ::= <digit> | 'B' | 'C' | 'D' | 'F' | 'H' |
'T' | 'U' | 'X' | 'Z'

<reserved condition> ::='A' | 'E' | 'G' | 'I' | 'J' | 'M' |
'O' | 'Q' | 'S' | 'V' | 'W' | 'Y'

0 - Drive input low
1 - Drive input high
2-9 - Drive input to super voltage #2-9
B - Buried Register preload
C - Drive input low, high, low
D - Drive input low, fast transition
F - Float input or output
H - Test output high
K - Drive input high, low, high
L - Test output low
N - Power pins, outputs not tested, and non- connected device pins
P - Preload registers
R - Read and retain register state
T - Observe registers
U - Drive input high, fast transition
X - Output not tested, input default level
Z - Test input or output for high impedance

**Table 2. Test Conditions**

### 7.2 Default Test Condition (X)

The X field defines the input logic level for test vectors not explicitly defined and for the "don't care" test condition. The X field will set test vectors 1 through the maximum (set by QV) to the default input test condition. If the X field is used, it must be specified after the QV and QP fields and before the first test vector.

Example

X1* Set default test condition to 1

In the following example vectors 2 and 5 would default to the don't care value of 0 and no outputs would be tested for vectors 2 and 5.

Example

```
QV5*
QP20*
X0*
V0001 101010000N0ZLLHHZ11N*
V0003 111XXXXXXN0ZHHLLZ11N*
V0004 011XXXXXXN0ZLHLHZ11N*
```

### 7.3 Test Vectors

Each test vector contains N test conditions where N is the number of pins on the device. Table 2 lists the conditions that can be specified for device pins.

The V field starts with a decimal vector number, followed by a space, then by a series of test conditions for each pin, and terminated by an asterisk. The vector number may include leading zeros.

Example:

```
V0001 000000XXXNXXXHHHLXXN*
V0002 010000XXXNXXXHHHLXXN*
V0003 100000XXXNXXXHHHLXXN*
V0004 110000XXXNXXXHHHLXXN*
```

The vectors are applied in numerical order to the device being tested. The highest numbered vector to be applied is defined by the QV field. If a vector is not specified during a data transfer the default value or a vector from a previous transfer will be used. If the same numbered vector is specified more than once, the data in the last vector replaces any data contained in previous vectors with that number. This allows the set of test vectors to be modified or "patched" without transferring the entire set.

### 7.4 Pin Sequence

The conditions contained in test vectors are applied to the device pins in numerical order from left to right unless specified otherwise with the P field. (The leftmost condition is applied to pin 1, and the rightmost condition is applied to pin 20 of a 20 pin device, for example. The timing sequence is not defined; a test condition may be applied to pin 5 before or after pin 4.) The P field indicates an alternative correspondence between the test conditions and the pin numbers. Each package pin, including nonconnects, must be represented by a number in the P field.

Example:

```
P 1 2 3 4 5 6 14 15 16 17 7 8 9 10 11 12 13 18 19 20*

V0001 111000HLHHNNNNNNNNNN*
V0002 100000HHHLNNNNNNNNNN*
```

Vector 1 will apply 111000 to pins 1 through 6 and HLHH to pins 14 through 17. Pins 7 through 13 and 18 through 20 are not tested (N).

### 7.5 Test Conditions

The test condition logic levels are defined by the device technology (e.g. TTL, CMOS, ECL). The 0 and 1 test conditions apply a steady state logic level to the device pin. The device tester should allow the applied input conditions to be overridden by bidirectional (input/output) device pins. The X or don't care test condition applies the default level defined by the X field. The F test condition applies a high impedance to the device pin.

The sequence that the input conditions are applied to the device is not defined, so multiple vectors should be used when the sequence is important. The following example ensures that pin 4 transitions to a logic level 1 before pin 3.

```
V01 XX00XXXXXNXXXXXXXXXN*
V02 XX01XXXXXNXXXXXXXXXN*
V03 XX11XXXXXNXXXXXXXXXN*
```

The test conditions 2 through 9 apply a non standard or super voltage to the device. This may be used to access special test modes. The levels are defined for each device and test vectors utilizing super voltages could damage "second source" devices.

The C test condition applies a logic level 0 until all other inputs are stable (and device timing specifications are met) then switches to a logic level 1 and returns to a logic level 0 before the outputs are tested. The K test condition goes from 1 to 0 to 1 in a similar manner. For devices more than one clock input, multiple test vectors should be used to ensure the proper clocking sequence.

The U test condition applies a logic level 0 until all other inputs are stable and internal set-up times met, then switches to a logic level 1 and remains at that level. This test condition should be used for any clock input that must make a single 0 to 1 transition. It is differentiated from the '1' test condition in that the device tester does not allow the input condition to be overridden by bidirectional device pins, thus allowing the U test condition to make a much faster transition. The D test condition is analogous to the U test condition except it applies a logic level 1 until all other inputs are stable and internal set-up times met, then switches to a logic level 0 and remains at that level.

The N test condition is used for power pins, output pins not tested, and non connected device pins.

After all inputs have stabilized, including clock, the output test are performed. The L test for a logic level 0 and the H test for a logic level 1.

The Z test condition test that an output is in a high impedance condition.

### 7.6 Register Preload

Register Preload means forcing or "jam loading" a register to a known state. Three types of register preloading are defined: "in-circuit", "output register", and "buried register".

"In-circuit" preload is accomplished with dedicated input pins and/or internal control logic and uses normal in-circuit logic levels. The standard input and clock test conditions may be used to preload the registers in these devices. The "output register" and "buried register" preload operations use non standard levels or "super voltages" to access special modes to preload the registers.

Because testing algorithms are unique for each device, the following generic methods may allow one set of test vectors to work with "second source" devices. The device programmer/tester will apply the specific superalgorithm for each device type.

### 7.6.1 P Preload Vector

A P preload vector is used to preload PLDs with output registers connected to device pins. The P preload vector is used to set the pins to a desired state.

The P symbol is applied to the clock pins controlling the registers. The vector values corresponding to the output registers must be 0 or 1. The value specified in the P vector is the value that is desired at the output device pins after the preload operation.

Example 1. Preload a 16R4 with the P preload symbol.

```
          11111111112     <- position in the
     12345678901234567890     vector
V0001 PXXXXXXXXXNXXX1101XXN*     <- P preload
                                    vector
V0002 0XXXXXXXXXN0XXHHLHXXN*     <- check
                                    loaded value
```

The PLD programmer determines the value to load into the register to achieve the desired result. In the example, the user wants a HIGH (or 1) to appear at the output pin. The programmer must actually load a 0 into the register because of the inverting buffer in front of the 16R4 register.

The PLD programmer must set up the appropriate input and clock conditions for a preload vector to work properly. For example, the outputs must be disabled for a 16R4 device by specifying a 1 for the active LOW enable pin in the preload vector. This is part of the preload algorithm that must be supplied by the PLD manufacturer. The designer must specify Xs (don't cares) for any unused input pins in the preload vector. If a symbol other than X is specified, the value required by the device algorithm in the PLD programmer will take precedence.

For devices with different clock pins controlling separate banks of registers, a P symbol must be applied to the appropriate clock pin to preload the corresponding register bank.

The P preload vector will use the pin sequence information specified by the QP and P fields. The position of the preload value in a P preload vector and the pin sequence will determine which register to preload.

Example 2: Preloading a 16R8 and using a second non-clocked vector to test the values preloaded.

```
V0001 PXXXXXXXXXNX11110000N*
V0002 0XXXXXXXXXN0HHHHLLLLN*
```

The values loaded into the registers are the values desired at the output pins after the preload operation.

Example 3: Programmer will override any non-X values on input pins in the preload vector. Device is a 16R8 with active-LOW enable control on pin 11.

```
V0001 PXXXXXXXXXN011110000N*
V0002 0XXXXXXXXXN0HHHHLLLLN*
```

The PLD programmer will override the active LOW enable signal (0) on pin 11 in vector V0001 to perform the preload.

### 7.6.2 B Preload Vector

The "buried register" preload method can be used for devices with internal registers not connected to device pins. This may also be used for registers connected to device pins.

A B preload vector will specify the binary value to be loaded into a register, without regard to the polarity or logic configuration of the device (i.e., active HIGH or LOW). If the register is connected to an output pin, then the value detected at the output pin will depend on any logic circuitry positioned between the register and the pin.

A B preload vector has the format:

Vxxxx BHbbbbbbbbbbbbbbbbbb* <- 20-pin PLD

The B preload vector numbered Vxxxx (in decimal) will be a vector containing the symbols 0-9 and A-Y. The B preload test vector length is equal to the number of pins in the device. The vector is terminated by an asterisk.

The first vector element from the left will be the B test condition symbol. The second element is an alphanumeric character used to identify the register group to preload. Register groups will be numbered from 0-9, A-Y. The character Z is reserved for future use.

The remaining test conditions in the vector refer to the states of uo to N-2 registers in the group identified by the group number, where N is the number of pins on the device. The allowed test conditions are "0", "1", "R", & "N".

The R test conditions is used by the programmer to read and retain the state of any register.

For example, if there are 8 registers in a 20-pin PLD, then these 8 registers are assigned to group number 0. Within group 0, the 8 registers will be assigned a unique number starting from 1. This register number is used to calculate the register's position in the preload vector.

V0001 B011110000NNNNNNNNN* <- 20-pin PLD

The test vector in the above example is used to preload 8 registers in a 20-pin PLD. Note that these registers can be input, output, or internal registers. The next vector elements after the B and group number refer to the states to be preloaded into the registers. If there are more vector positions than registers in the PLD, then the rest of the vector is filled with the N symbols.

Example 4: Preloading a device with 12 internal registers. All 12 registers can be defined in one B preload vector.

V0001 B01110001010NNNNNN*

Example 5: Preloading the last 6 registers in a 20-pin PLD with 26 internal registers.

V0001 B1RR010100NNNNNNNNN*

Registers 19-26 are contained in register group 1. The status of the 19th and 20th registers is retained by using the R test condition.

Example 6: Preload registers 17 to 23 on a 20-pin device with 26 registers.

V0001 B0RRRRRRRRRRRRRRR10*
V0002 B111101RRRNNNNNNNNN*

Vector V0001 will preload registers 17-18 (in group 0), and V0002 will preload registers 19-23 (group 1). The other registers will remain unchanged.

The group number for the Xth register and the position of the Xth register within a preload vector can be calculated by the following formulas:

GROUP_NO = (X-1) div (PINS-2)

POSITION_IN_VECTOR = X - ((PINS-2)*GROUP_NO) + 2

where: PINS = # of device pins
div = integer division (fractions are truncated)

For example, if a 20-pin PLD has 40 registers, then the 31st register will belong in group 1.

GROUP_NO = (31-1) div (20-2)
= 30 div 18
= 1

POSITION_IN_VECTOR = 31 - (18*1) + 2
= 31 - 18 + 2
= 15

V0001 B1RRRRRRRRRRRR1RRRRR*
^ ^
^ register 31 in vector
group number 1 position 15

When a B preload vector is used in a JEDEC file, the QP field must specify all the device pins. The P (pin order sequence) will not affect the B preload vector.

For both the B and P preload vectors, 0 and 1 should be used to specify register preload values. The outputs are tested using the L and H symbols. To verify the result of a preload vector (either B or P), use a separate test vector or sequence of vectors to test the result of a preload operation.

### 7.7 Register observation

Register Observation means viewing the value of a register in a PLD. Output, internal, and input registers can have their contents examined, as provided for by special access circuitry on a device. For internal registers in particular, three types of observation are defined: "product term", "super-voltage", and "in circuit" observation.

Product term observation, if provided on the device, is accomplished with dedicated product terms and uses normal in-circuit logic levels. The standard input conditions may be used to observe the internal registers in these devices if the observability product terms have been so configured by the user. The "super-voltage" observation operation uses non standard levels or "super-voltages" to access special modes to observe the internal registers.

Because algorithms are unique for each device, the following generic methods will allow one set of test vectors to work with "second-source" devices. The device programmer/tester will apply the specific algorithm for each device type. A T vector will specify the binary value to be tested on a register, without regard to the polarity or logic configuration of the device (i.e., active HIGH or LOW).

A T vector has the format:

Vxxxx THbbbbbbbbbbbbbbbbbb* <- 20-pin PLD

Syntax of T vector:

<group number> ::= <digit> | 'A' | 'B' | .. | 'Y' | 'Z'

<observation test vector> ::= 'V' <number> <delimiter>
'T' <group number> <test conditions> :N-2 '*'

The T vector numbered Vxxxx (in decimal) will be a vector containing the symbols 0-9 and A-Y. The T test vector length is equal to the number of pins in the device. The vector is terminated by an asterisk.

The first vector element from the left will be the T test condition symbol. The second element is an alphanumeric character used to identify the register group to observe. Register groups will be numbered from 0-9, A-Y. The character Z is reserved for future use.

The remaining test conditions in the vector refer to the states of up to N-2 registers in the group identified by the group number, where N is the number of pins on the device. The allowed test conditions are 'L', 'H', 'X', & 'N'.

For example, if there are 8 registers in a 20-pin PLD, then these 8 registers are assigned to group number 0. Within group 0, the 8 registers will be assigned a unique number starting from 1. This register number is used to calculate the register's position in the observation vector.

V0001 TLHHHHLLLLNNNNNNNNNN* <- 20-pin PLD

The test vector in the above example is used to observe 8 registers in a 20-pin PLD. Note that these registers can be input, output, or internal registers. The next vector elements after the T and group number refer to the expected states to be tested on the registers. If some registers are not to be tested, an X (or don't care) symbol should be used. If there are more vector positions than registers in the PLD, then the rest of the vector is filled with the N symbol.

Example 1: Observing a device with 12 registers. All 12 registers can be tested in one T vector.

V0001 T0HHHLLLHLHLHLNNNNNN*

Example 2: Observing the last 6 registers in a 20-pin PLD with 26 registers.

V0001 T1XXLHLHLLNNNNNNNNNN*

The status of the 19th and 20th registers is ignored by using the X test condition. The last 6 registers are contained in register group 1.

Example 3: Test registers 17 to 23 on a 20-pin device with 26 registers.

V0001 T0XXXXXXXXXXXXXXXXXXHL*
V0002 T1HHHLHXXXNNNNNNNNNN*

Vector V0001 will observe registers 17-18 (in group 0), and V0002 will observe registers 19-23 (group 1). The other registers will be ignored.

The group number for the Xth register and the position of the Xth register within an observation vector can be calculated by the following formulas:

GROUP_NO = (X-1) div (PINS-2)

POSITION_IN_VECTOR = X - ((PINS-2)*GROUP_NO) + 2

where: PINS = # of device pins
div = integer division (fractions are truncated)

For example, if a 20-pin PLD has 40 registers, then the 31st register will belong in group 1.

GROUP_NO = (31-1) div (20-2)
= 30 div 18
= 1

POSITION_IN_VECTOR = 31 - (18*1) + 2
= 31 - 18 + 2
= 15

V0001 T1XXXXXXXXXXXXX1XXXXX*
^ ^
^ register 31 in vector
group number 1 position 15

When a T vector is used in a JEDEC file, the QP field must specify all the device pins. The P (pin order sequence) will not affect the T vector.

For the T vector, L and H should be used to specify register test values. The registers are preloaded using the 0 and 1 symbols in a B or P preload vector.

## 8 PROGRAMMER/TESTER OPTIONS

### 8.1 Security Fuse (G)

The security fuse(s) of certain logic devices may be enabled for programming by sending a 1 in the G field. The security fuse prevents the reading of the fuse states.

Syntax for the Security Fuse Field:

<security fuse> ::= 'G' <binary-digit> '*'

Example:

G1* Enable security fuse programming.

### 8.2 Signature Analysis Test (S, R, T)

Signature Analysis tests are specified by the S, R, and T fields. The S field defines the starting vector for the test. The possible states are 0 and 1. The R field contains the resulting vector or test-sum. The T field denotes the number of test cycles to be run.

Syntax for Signature Analysis Test:

<starting vector> ::= 'S' <test condition>:N '*'
<resulting vector> ::= 'R' <hex-digit>:8 '*'
<test cycles> ::= 'T' <number> '*'

N ::= number of pins on device

Example:

S0100010000111000111110110*
R5BCD34A7*
T01*

### 8.3 Access Time (A)

The A field defines the propagation delay for test vectors in one nanosecond increments. This field may include optional subfields.

Syntax for Access Time

<access time> ::= 'A'{<field characters>} <number> '*'

Example:

A25*
APD25*

## 9 EXAMPLES

### 9.1 Data File Examples

<STX>
*
L0000
11111011111111111111111111111101111111111111111111111111
11101111111111111111111111111000000000000000000000000000
01010111011110111111111111111000000000000000000000000000
00000000000000000000000000000000000000000000000000000000
01010111101110111111111111110000000000000000000000000000
00000000000000000000000000000000000000000000000000000000
01010111011101111111111111110000000000000000000000000000
00000000000000000000000000000000000000000000000000000000*
<ETX>5718

**Example 1. Minimum file for device programmer as defined by Jedec Standard No. 3-A, October 1983.**

```
File for PLD 12S8    Created on 8-Feb-85  3:05PM
6809 memory decode  123-0017-001
Joe Engineer  Advanced Logic Corp *
QF0448*
F0*
L000 11111011111111111111111111111*
L028 10111111111111111111111111111*
L056 11101111111111111111111111111*
L112 01010111011110111111111111111*
L224 01010111101110111111111111111*
L336 01010111011101111111111111111*
C124E*
```

**Example 2. Data file for device programming.**

```
File for PLD 12S8    Created on 8-Feb-85  3:05PM
6809 memory decode  123-0017-001
Joe Engineer  Advanced Logic Corp *
QP20* QV8*
V0001 000000XXXNXXXHHHLXXN*
V0002 010000XXXNXXXHHHLXXN*
V0003 100000XXXNXXXHHHLXXN*
V0004 110000XXXNXXXHHHLXXN*
V0005 111000XXXNXXXHLHHXXN*
V0006 111010XXXNXXXHHHHXXN*
V0007 111100XXXNXXXHHLHXXN*
V0008 111110XXXNXXXLHHHXXN*
```

**Example 3. Data File for device testing.**

```
File for PLD 12S8    Created on 8-Feb-85  3:05PM
6809 memory decode  123-0017-001
Joe Engineer  Advanced Logic Corp *
QP20*          N Number of pins*
QF0448*        N Number of fuses*
QV8*           N Number of vectors*
G1*            N Program security fuse*
F0*            N Default fuse state*
X0*            N Default test condition*

N Fuse RAM Data*
L0000
11111011111111111111111111111
10111111111111111111111111111
11101111111111111111111111111*
L0112
01010111011110111111111111111*
L0224
01010111101110111111111111111*
L0336
01010111011101111111111111111*
```

```
N Test Vectors*
V0001 000000XXXNXXXHHHLXXN*
V0002 010000XXXNXXXHHHLXXN*
V0003 100000XXXNXXXHHHLXXN*
V0004 110000XXXNXXXHHHLXXN*
V0005 111000XXXNXXXHLHHXXN*
V0006 111010XXXNXXXHHHHXXN*
V0007 111100XXXNXXXHHLHXXN*
V0008 111110XXXNXXXLHHHXXN*

N Fuse RAM checksum*
C124E*

N Signature Analysis test  information*
T01*
S00000000000000000000*
R95E4B822*
```

**Example 4. Data File for programming and testing with options.**

```
File for PLD 12S8    Created on 8-Feb-85  3:05PM
6809 memory decode  123-0017-001
Joe Engineer  Advanced Logic Corp *
QP20* QF448* QV8*
F0*
V1 000000000N000HHHL00N*
V2 010000000N000HHHL00N*
V3 100000000N000HHHL00N*
V4 110000000N000HHHL00N*
L0  11111011111111111111111111111*
L28  10111111111111111111111111111*
L56  11101111111111111111111111111*
L84  00000000000000000000000000000*
L112 01010111011110111111111111111*
L224 01010111101110111111111111111*
L336 01010111011101111111111111111*
L140 11111111111111111111111111111*
L140 00000000000000000000000000000*
C124E*
V8 111111111N111HHHHL11N*
V6 111010000N000HHHH00N*
V7 111100000N000HHLH00N*
V5 111000000N000HLHH00N*
V8 111110000N000LHHH00N*
```

**Example 5. Data file showing position independence of fields.**

# ANNEX

## A1 INTERACTIVE PROGRAMMING ADDENDUM TO JEDEC STANDARD NO. 3B

### A1.1 INTRODUCTION

#### A1.1.1 Purpose and Scope

This addendum was developed to provide limited remote, or microcomputer host, control of PLD programming equipment. This mode of operation is required when the final PLD programming pattern cannot be determined until after the user defines the functional requirements of the PLD and, in some cases, the device is programmed and functionally tested.

While in this mode certain information files are bidirectionally transferred between a host microcomputer and the programming equipment. Receipt of JEDEC files are always acknowledged by the programming equipment and detectable error device failures are always reported to the microcomputer. The use of supervoltage fields and vectors are also permitted in continuation files while in the interactive programming mode.

It is not the intent of this addendum to impose a system configuration that requires a host computer to program a device. All proposed fields are optional and the manufacturer of programming equipment may elect to provide the function of the host computer within its programming equipment.

The intent of this addendum is to provide the described functions until superseded by a more comprehensive standard supporting computer remote control of programming equipment.

#### A1.1.2 Summary of Reporting and Testing Fields

The reporting and testing information is contained in various fields. The following list gives the field identifier and description of those required of this addendum but not defined in 1.2 of standard 3A. Detailed descriptions of these identifiers are given in section 3 .

| Identifier | Description |
|---|---|
| M | Interactive mode |
| ME | Error message |
| MV | Failed Test Vector |
| QE | Super-voltage definition |

#### A1.1.3 Changes to Standard No. 3A

This addendum defines the supporting fields required for an optional mode of operation for PLD programming equipment. This mode of operation is referred to as the interactive programming mode. All files used in this mode of operation must be contained within <stx> and <etx> characters, end with an xsum (transmission checksum), and follow the applicable rules, transmission protocol, and syntax of standard No. 3A except as specifically noted.

## A1.2 FILE TYPES

### A1.2.1 General

The following list contains the three basic file types encountered while in this mode of operation. The A-file is a programming equipment to computer transmission file while the other files are computer to programmer transmissions.

| type | originates | function |
|---|---|---|
| S-file | computer | Source |
| A-file | programmer | Acknowledge |
| C-file | computer | Continuation |

### A1.2.2 Source file (S-file)

This file type is a programming download file formatted to standard No. 3A. An S-file always includes an Mn* field with n=0 identifying the file as a source file indicating a new PLD programming/test session and telling the programmer to purge memory of data from any previous PLD program/test session. The M0* field precedes the Ln data* fields and the optional test vectors. The programming equipment must always respond to a source file with an A-file. The S-file is the initial file if C-files are used and tells the programmer that another file may follow. The S-file will be in one of the formats listed below.

| Sa | Sb |
|---|---|
| <stx> | <stx> |
| M0* | M0* |
| Ln data* | Ln data* |
| <etx> | Vn vector* |
| xsum | <etx> |
| xsum | |

Note: A vector may contain one or more super-voltage defining input characters. Each character is an integer between 2 and 9 instead of the most often used 0 or 1. Integers greater than 1 refer to a super-voltage and tells the programming equipment to apply a super-voltage of the assigned value (the value of which was previously defined by the manufacturers individual device programming specification) to the designated device package pin. This type of vector defines which super-voltages are used, and to which device pins the voltages are applied.

Format Sa is a simple JEDEC formatted file with the additional M0* field telling the programming equipment to enter and/or operate in the interactive programming mode.

Note: Some programming equipment may require manual instruction to enter the interactive programming mode. While in this mode all files must adhere to the applicable rules of standard No. 3A and this addendum.

Format Sb is the same as a Sa-file except for the test vectors. The vectors may contain super-voltage data.
The following example is a Sb-file where super-voltage #2 has been previously defined in the PLD manufacturers programming specification. This file contains standard test vectors and a single Super-Voltage vector (V4).

Example:

```
<stx>* M0* QP20* QF392* F0*
L000
11111111111111010111101111*
L096
11111111101101010111010111*
L144
11111111101010111111111111*
L192
11101111111111111111111111
01110111111111111111111111
11111111101101011111111111*
L288
11111111111111010111111101*
QV5* X0*
V1 000XXXXXXXNXXNNHNNNXN*
V2 111XXXXXXXNXXNNHNNNXN*
V3 101XXXXXXXNXXNNLNNNXN*
V4 XXX2XXXXXXNXXNNNLNNXN*
V5 XXX0X11XXNXXNHNNNNXN*
C134A* <etx>xsum
```

Note: S-files vectors with super-voltage specifying characters can only be used where the super-voltage(s) have been pre defined in the PLD manufacturers individual device programming specification.

After accepting the S-file the programmer will

a) program the device per the Ln data* field data,
b) test per any vector fields,
c) always respond with an A-file with errors reported, and
d) be ready to receive a C-file or another S-file

### A1.2.3 Acknowledgement file (A-file)

This file type is provided by the programmer following receipt of each S-file or C-file and includes either the M1*, MEn*, or MVn vector* fields. The file will be in one of the four formats shown in the following table.

| Aa | Ab | Ac | Ad |
|---|---|---|---|
| <stx> | <stx> | <stx> | <stx> |
| M1* | MEn* | MVn vector* | MEn* |
| <etx> | <etx> | <etx> | MVn vector* |
| xsum | xsum | xsum | <etx> |
| xsum | | | |

Format Aa tells the controlling microcomputer that the S-file or C-file was received and processed with no detectable errors.

Format Ab tells the microcomputer of an error occurrence while loading or processing the S-file or C-file. The MEn* field signals the errors with n defining the error types. Applicable errors can be defined with as many ME fields as required. The error types are listed in 3.3.

Format Ac is used if test vectors are included in the downloaded file and vector failures have occurred. Upon receipt of this file the microcomputer will download a correction, or programming, continuation file (Cc-file or Cd-file), the original or a new S- file for the next PLD, or quit.

If failures occurred on pin 15 of vectors 2 and 4 in the example Sb-file shown in 2.2, the following Ac-file would be sent to the host. Note: Vector 4 dictates that a super-voltage be applied to device pin 4.

Example:

<stx>
MV2 111XXXXXXXNXXNNLNNNXN*
MV4 XXX2XXXXXXNXXNNHNNNXN*
<etx> xsum

Format Ad tells the host that other potentially fatal errors have occurred in addition to test vector failures. The host may be able to take corrective action with a C-file if the errors were limited to fuse programming or verification failures whose locations may be determined by the failure pattern indicated by the failed vectors.

Note: The least complex use of the interactive programming mode provides a simple computer controlled device programming system. This procedure does not make use of super-voltages, i.e., C-files or QE fields and is as follows:

1) Sa-file (Sb if vectors in file) to programmer
2) Aa, Ab, or Ac-file to microcomputer
3) microcomputer displays applicable message, ie. error, request for new filename, etc.
4) microcomputer accepts applicable inputs, ie. new filename, requests to repeat file, etc.
5) go to step 1 if additional devices to be programmed
6) software exit

### A1.2.4 Continuation file (C-file)

This file type is software generated, as required for each individual PLD, and is not available or usable for any other PLDs. The C-file always includes an Mn* field where n=2 indicating a continuation file and telling the programming equipment to hold all relevant PLD data and to permit patching of PLD data as required.

A continuation file may contain a QEn value*, Ln data*, and/or Vn vector* fields, but need not repeat the QP, QF, and F fields of the most recent S-file. This file type is always preceded by an S-file and is the only file type permitted to contain QEn value* fields. The file will be in one of the four formats shown in the following table.
Note: For selected PLDs, on-chip voltage sources can be programmatically altered or defective logic paths can be replaced. Ln data* fields, created by the computer and included as part of the C-files, define the new values or logic paths. The contents of the Ln data* fields can be algorithmically defined as a result of the user defined logic in conjunction with the pattern of vector failures.

| Ca | Cb | Cc | Cd |
|---|---|---|---|
| <stx> | <stx> | <stx> | <stx> |
| M2* | M2* | M2* | M2* |
| QEn value* | QEn value* | Ln data* | QEn value* |
| Vn vector* | <etx> | <etx> | Ln data* |
| <etx> | xsum | xsum | Vn vector* |
| xsum | <etx> | | |
| xsum | | | |

Note: Vectors in a C-file may contain super-voltage references.
Format Ca tells the programming equipment to set the values of one, or more, super-voltages as directed by the QEn value* field(s) and to perform the tests per the applicable Vn vector* field(s). Values defined in the QE fields supersede any predefined super-voltage values.

Note: In some cases, on-chip voltage sources can be measured, internal to the the device, by comparing their values to external voltage references (the super-voltages) applied to the device pins during certain vector tests. The QEn value* fields define the value of the external voltage references. The vectors define which super-voltages are used, and to which device pin the voltages are provided. The result of the on-chip voltage comparison is seen as logic level device outputs and referenced in the vectors as H or L output level identifying characters.

Devices may fail one or more test vectors because predefined internal voltage references do not agree with internal references developed as a result of device programming.

Format Cb permits changing the values of the super-voltage and performing tests with predefined vectors (those contained in a source file or earlier continuation file).

Format Cc is used to change the value of on-chip voltage sources by programming selected cells. This is the required format when the default values of the super-voltages are defined in the programming specification, tested with a Sb-file, and one-pass correction is acceptable. This file type is also used when the super-voltages have been previously defined in an earlier C-file within the current programming/testing session (while programming or testing the same PLD).

Following is an example Cc-file transmitted in response to the Ac-file shown in 2.3. The vectors that need be retested after the cells are programmed need not be repeated in this file. All vectors and other applicable PLD data previously transmitted are resident in the programmer until a new S-file is provided. Cell numbers 384 through 391 in this file are reserved to alter internal voltage references. The source Sb-file for this example is shown in 2.2. Note that the C (fuse checksum) has been updated to reflect the changes in the fuse map.

Example:

<stx> M2*
L384
11010010*
C13AB* <etx>xsum

Format Cd changes the values of on-chip and super-voltage sources and defines one or more appropriate vectors to test the results.

Note: The least complex use of the QE fields in the interactive programming mode is as follows:

1) Sc-file (Sd if non-super-voltage defining vectors are required) to programmer
2) A-file to microcomputer
3) if no vector fails then go to step 6
4) Cc-file to programmer
5) A-file to microcomputer
6) message and/or operator inputs at microcomputer
7) go to step 1 if additional devices to be programmed
8) software exit

This provides a simple computer controlled device programming system with a one-pass correction of on-chip voltage sources.

## A1.3 FILE REPORTING FIELDS

### A1.3.1 General

Two major field identifiers are used in the implementation of this addendum. The Q field identifier is defined in standard No. 3A. The M field identifier is new to this addendum.

<field identifier> ::= 'M' | 'Q'

Field Identifiers

M - mode Q - value (per standard 3A)

### A1.3.2 Mode declaration field (Mn)

The Mn* field in host provided JEDEC files instructs the programming equipment to enter and/or continue operation in the interactive (LOAD, GO, and REPORT ERROR) programming mode. The M0 field is contained in all S-files, M1 in A-files where no errors are being reported, and M2 in all C-files.

### A1.3.3 Error reporting (MEn)

This field is found only in A-files reporting errors detected by the programming equipment. More than one MEn field may be in the file if more than one error type is detected with n indicating the error type.

The MEn* field is required if a fatal, or potentially fatal, error or failure occurs. The host microcomputer provides the user with the appropriate message and takes the appropriate action if applicable when the cause of an error is identified. The response to PLD programming failures is based in some cases on user inputs. The final action taken may be to end the session, initiate corrective action with a C-file, or to begin a new PLD programming session with an S-file.

While in this mode of operation the programming equipment will continue processing the file after certain error types occur as defined by the PLD manufacturers programming specification. If a programming, verification, or vector failure were to occur, the equipment would continue the programming, verification, vector testing, and report all detected error types to the host microcomputer. The final determination to abort the program/verification/test session is made by the microcomputer or by error limits as defined in the PLD programming specification.

Error message field (MEn)

<error message> ::= 'ME' <number> '*'

Error messages:

field error type
ME0* undefined error
ME1* no device in socket
ME2* device insertion error
ME3* reversed device
ME4* device over-current fault
ME5* faulty device
ME6* electronic ID verify error
ME7* load or file error
ME8* secured device
ME9* security fuse programming error
ME10*fuse checksum
ME11*transmission checksum
ME12*programming failure
ME13*verify failure
ME14*cell pre-programmed to wrong state
ME15*preload not supported by device
ME16*test vector syntax error
ME17*super-voltage definition error
ME18*unrecoverable error

Note: If a particular error type is not defined the ME0* field may be used. Certain error types such as ME12, ME13, and ME14 need not necessarily be interpreted as fatal by the host microcomputer. The device programming specification may specify limits for certain error types and expect an ME18* field if the number of programming failures, normally reported by an ME12* field, exceeded the limit.

#### A1.3.4 Vector failure field (MVn)

This field is found only in A-files reporting test vector failures to the host. All vector failures are reported in the same sequence as listed in the source file.

One MVn vector* field exists for each failed vector with n indicating the vector number. An MVn vector* field indicates a failed vector. The vector number (n) and input data are the same as that most recently provided by the computer in the S-file or C-file. The vector data reflects the data as seen by the programming equipment allowing the computer to compare the failed vectors with the original vectors to locate the errors and take the appropriate action.

### A1.4 DEVICE TESTING FIELDS

#### A1.4.1 General

The use of QE* fields and associated vectors (with super-voltage data) permit the measurement of on-chip voltage references. The actual value of the voltages may be determined, in part, by the user defined logic, or digital data, programmed into the device. A separate QE field is required for each super-voltage to be defined in the vectors, and all must occur in the file prior to the associated test vectors.

### A1.4.2 Super-Voltage Definition (QEn)

The use of the QE field may provide information regarding device characteristics. Many new devices allow for, and require that, these device characteristics be altered if they are not satisfactory. This procedure will require that additional JEDEC files be acted upon by the programmer depending on the results of, and in a manner dictated by, the previous file. This procedure can only be accomplished in a C-file after the recognition of a M0* field in the original JEDEC formatted file (S-file). Under no circumstances may a source file contain a QE field.

Syntax of QE field:

<super-voltage> ::= 'QE' <digit> <delimiter> <number> '*'

Example:

QE3 10500* Indicates Super-Voltage #3 is 10.5 volts
QE5 64300* Indicates Super-Voltage #5 is -14.3 volts

Definition

Each QEn d* field, in conjunction with proper Vn vector* fields, specifies the value of a super-voltage used within one, or more, of those vectors.

n is an integer 2 to 9 representing one of the eight super-voltages which may be defined.
d is a decimal number from 0 to 99999 (1-5 digits) defining the value, in millivolts, of the super-voltage. The actual voltage is d mod 50000, permitting values from -49.999 volts to +49.999 volts to be defined. The numbers from 0 to 49999 define the positive voltages.

## A1.5 IMPLEMENTATION CONSIDERATIONS

### A1.5.1 Standard No. 3A

Standard No. 3A describes a means of patching the fuse list portions of files in programmer memory. To ensure that new files completely replace earlier files in the interactive programming mode the programmer must recognize the M0* as being in a new (source) file and as a signal to clear memory.

All source files must adhere strictly to the standard. Continuation (C-files) need not repeat previously transmitted PLD data to the programmer.

### A1.5.2 Programming Specifications

The use of a QE field requires that the device manufacturers programming specification include the number of super-voltages and the following data, if applicable, for each:

a) identification number (2 through 9)
b) default value (lower limit if not defined)
c) upper and lower limit
d) slew rate limits of each edge
e) minimum hold time to vector execution
f) device pin to which it is applied
g) the source/sink current required
h) the cell numbers affecting QE values

Comment: The following limitations may restrict the use QE fields for some programming equipment:

1) number of programmable voltage sources
2) device pins to which the voltages may be applied
3) the accuracy and resolution of the voltage sources

Note: the source/sink current requirements of the target pin for a super-voltage may have a dramatic effect on the accuracy and resolution of the super-voltage.

### A1.5.3 Programming Equipment QE field support requirements

For device programmers supporting this field, the recognition or acceptance of a QE field, or a vector using a super-voltage, would be restricted to supported devices and the use detailed in the applicable device manufacturer's programming specification. The device programmer must check QE values and compare to specified limits.

The device programmer should ensure that file type is appropriate for the device/pinout code and check super-voltage specifying vector locations and compare to the specified pins.

## A1.6 EXAMPLE FILES

### A1.6.1 Voltage reference adjustment

These examples represent a single PLD programming session where the values of a number of on-chip voltage sources are changed.

```
<stx>* M0* QP20* QF2056* F0*
L0000
11111111111111111111111111111111
10111111110111111111101010011101
L1024
10111111110111111111111111101111
11111111110111111111101111101111*
L1792
11111111111111111111111111111111
10111111111111111111111011111110*
QV5* X0*
V1 000XXXXXXXNXXNNHNNNXN*
V2 100XXXXXXXNXXNNLNNNXN*
V3 111XXXXXXXNXXNNHNNNXN*
V4 101XXXXXXXNXXNNLNNNXN*
Ccsum* <etx>xsum
```

Example 1. An Sb-file with vectors not dictating a super-voltage where an internal voltage reference will be affected, in part, by the logic pattern defined by the L fields. Programmer memory is reserved for subsequent L and V fields. Previous PLD data is erased from memory.

<stx>M1* <etx>xsum

Example 2. An Aa-file from the programmer indicating no errors in previously transmitted Sb-file.

<stx>M2* QE2 10100* QE3 10300* QE4 10500*
V5 234XXXXXXNXXLLLLLLXN*
<etx>xsum

Example 3. A Ca-file with super-voltage assignments, based on the Sb-file previously transmitted. Any default super-voltage values that may have been specified by the PLD manufacturers device programming specification are replaced by the values specified in this file. The previously transmitted QV and X fields are held valid. The vector is patched into the programmer memory.

<stx>
MV5 234XXXXXXNXXLHLLHHXN*
<etx>xsum

Example 4. An Ac-file providing the data for a vector failure in the Sb-file.

<stx> M2*
L2048
00101110*
Ccsum* <etx>xsum

Example 5. The Cc-file alters the values of on-chip voltage references by programming additional fuses. The new values were determined by the pattern of vector failures. The pattern of vector failures was determined by the logic pattern programmed into the device and the original values of the on-chip references. The fuse checksum reflects the new fuse map. All resident vectors are tested after programming. The values of the super-voltages are unchanged.

<stx>M1* <etx>xsum

Example 6. The final Aa-file for the session indicating a correct programming of the on-chip voltage sources. A new PLD session is initiated by an S-file.

### A1.6.2 Logic replacement

This example represents a PLD programming session where redundant logic paths are programmed to replace existing, defective, paths.

```
<stx>* M0* QP20* QF2056* F0*
L0000
10111111110101010110101010011101
L1024
10111101101110101011111111101111
11111111110111111111101111101111*
L1792
11111111101111110111101111010111
10111111111111111111111011111110*
QV4* X0*
V1 000XX10XXNXXNNHLHNXN*
V2 100XX11XXNXXNNLLLNXN*
V3 111XX11XXNXXNNHHLNXN*
V4 101XX10XXNXXNNLLHNXN*
Ccsum* <etx>xsum
```

Example 1. An Sb-file with vectors (no super-voltage data). Since this is a source file, previous PLD data is erased from memory.

```
<stx>
MV1 000XX10XXNXXNNHLLNXN*
MV3 111XX11XXNXXNNHHHNXN*
<etx>xsum
```

Example 2. An Ac-file providing vector failure data (pin 18).

```
<stx> M2*
L2048
10101110*
Ccsum* <etx>xsum
```

Example 3. The Cc-file replaces the failed logic path by programming selected additional fuses. The replacement path was determined by the pattern of vector failures and the logic pattern programmed into the device by the source file. The fuse checksum reflects the new fuse map. Resident vectors are tested after programming. Had the logic re-routing cells been previously programmed a fuse checksum failure would have been given.

```
<stx>M1* <etx>xsum
```

Example 4. The final Aa-file for the session indicates correct programming and operation of the logic path replacement circuits.